Reference E: Standards for Year 2000 Conversion / Year 2000 Technical Panel (Version 3.0)

This document was originally published on January 16, 1997 and was updated on November 19, 1997. It has been reformatted and re-edited for grammar before inclusion into this ***Readiness Guide Reference Library***. Any technical change from version 2.0 is marked with a change bar.

Table Of Contents

Originally Published on January 16, 1997; Revised November 19, 1997

1.0 Purpose

This document provides a common set of standards and guidelines for Year 2000 renovation at HUD.

2.0 Scope and Assumptions

2.1 HUD-wide

The standards contained in this document apply to all of HUD. The Year 2000 Technical Panel shall review any proposed exceptions to the standards outlined in this standards document. Please forward requests for exception to the Team 2000 Project Office via cc:Mail to *team_2000@hud.gov*. Requests should contain the following information:

- Name of person making request
- Phone number
- System name
- Brief description of reason for request

Requests will be presented to the technical panel so that the panel can decide on granting the request and possibly making a permanent exception or change to the standards. The panel will track all exceptions.

Standards exist on various levels:

- Enterprise;
- Platform (Hitachi, UNISYS, PC/LAN);
- Application DBMS (DB2, DMS2200, FoxPro.);
- Language; and
- Application family. (A family of applications is defined as a set of programs that interfaces largely with one another. Applications that share a database are, by definition, an application family).

2.2 Assumptions

This document uses the words *shall* and *must* to indicate a firm standard and the word *should* to indicate a recommendation. This document uses the terms *application development team* and *renovation team* to indicate the body responsible for making decisions regarding—and actually conducting—Year 2000 conversion development on a system. This document uses the terms *application* and *system* interchangeably.

This document uses the term renovation to describe development work related to Year 2000 conversion of a system.

2.3 What Date Elements are Impacted by the Year 2000

Once you have identified all the date related elements in your system, you must determine which date related elements are used in comparisons, calculations, or sorts. These are date related elements that will likely require attention.

Other date related elements may be maintained in their current formats. An example of such an element is a date field that is used only to print to a report.

Once you have narrowed your focus to the date related elements that are used in comparisons, calculations, or sorts, you must decide upon your approach to solving your Year 2000 problem. See **Section 4**, ***Approaches to Renovation***.

3.0 HUD-wide Year 2000

3.1 Interagency Data Transfer

The Federal Government has mandated that dates in interagency data transfer will contain 4-digit years. HUD systems fall under this mandate. Existing systems shall retain their current data type when expanding to 4-digit years.

Since HUD may move forward with its renovation before other agencies, we must create bridge programs to sit between HUD applications that use 4-digit years and applications of other agencies that use 2-digit years. When the other agencies move to 4-digit years, we can remove our bridges.

The Year 2000 Project has initiated a plan to contact the agencies and other organizations and entities with whom HUD exchanges data electronically. This effort will establish a line of communication in order to coordinate our conversion efforts.

3.2 Intersystem Data Transfer

The preferred format for dates contained in intersystem data transfer is in 4-digit year format. When employing 4-digit years, the format should be:

Format	Data Type	Date Type
CCYYMMDD	Application Specific	Calendar date
CCYY	Application Specific	Year
CCYYMM	Application Specific	Year-month

Data type (whether the variable is defined as numeric, text, and so on.) shall be decided at the application level. Renovation teams shall maintain the current data type of a variable or field if possible. For example, if an application currently passes a 6-digit variable that is defined as numeric to another application, it shall continue to pass a numeric variable once it is expanded to 8 digits.

The renovation teams of both sending and receiving applications must work together to make decisions regarding expansion. No expansion of dates contained in inter-system communication will be valid unless both the sender and recipient agree upon this format. Each renovation team that expands any date fields must note record length changes in their ***External Entities Document***. This document should be part of their ***High Level System Binder*** for every file expanded. (For an example, refer to the High Level System Binder in the **HUD Year 2000 Implementation Strategy document**.) This document should have a field for pre-Year 2000 renovation file length and post renovation file length. There should also be a comment field for other changes. Renovation teams shall exchange these documents.

If sending and receiving systems decide upon date expansion as a communication standard, but one system renovates earlier than another, a bridge must be built in order to continue communication with that system. All systems will not be converted to year 2000 compliance simultaneously. Application renovation teams must not allow data exchange with a non-compliant system to keep them from making their system year 2000 compliant. If bridge building appears to be an overwhelming task, then the application development team should raise the issue with project management in order to initiate plans to cutover the related systems together.

However, if it is determined that an application can operate successfully with windowing then it is not necessary for the renovation team to expand dates contained in an inter-system data exchange. (Refer to **Section 4, Approaches for Renovation** for a definition of windowing.)

When not to expand dates in inter-system communication:

- If both the sending and receiving systems use 2-digit years internally, for example, through windowing techniques, the systems may exchange dates containing 2-digit years;
- If a date is not used for any computational or conditional purpose, for example, it is used simply for printing to a report, then the date may be passed in 2-digit year format; or
- If both the sending and receiving systems use 2-digit years externally and 4-digit years internally through windowing techniques.

The renovation teams of both sending and receiving applications must work together to make decisions regarding date expansion.

Note that if any dates in a record, database table, or file require expansion, the team might consider expanding all date fields in that element since the element needs to undergo conversion in any event.

See **Section 4**, **Approaches for Renovation**, for the benefits and drawbacks of each approach.

3.3 System Date

All programs on HUD's mainframe platform must be modified to retrieve the system date in the standard CCYYMMDD format. The method to do this varies from platform to platform.

Notes on TransCentury:

If you are performing sophisticated date handling using TransCentury, the package includes the U001 function to retrieve the system date in a variety of formats. On the other hand, if you are only retrieving the system date, the overhead of the TransCentury package (about 100K bytes) is too large to justify its use. Vendor supplied COBOL commands or subroutines can be used instead.

TransCentury is not installed for the Unisys ASCII COBOL compiler due to address space limitations.

For the following discussion, assume the following variable has been defined in working-storage:

```
01   Date-8        PIC X(8).
 *   Format is CCYYMMDD
```

3.3.1 Hitachi

The appropriate technique is a function of the compiler being used.

3.3.1.1 OS/VS COBOL

No alternatives have been offered by IBM. Support for this compiler was terminated years ago. No changes have been made to the compiler to support 4-digit years.

3.3.1.2 VS COBOL II

For non-CICS programs, IBM has provided a module to retrieve the Date-8 variable:

```
CALL 'IGZEDT4' USING Date-8.
```

CICS Version 4.1 has new FORMATTIME options which can retrieve the date in various Year 2000 compliant formats.

3.3.1.3 COBOL for OS/390

COBOL for OS/390 supports the new ANSI standard capabilities of Intrinsic Functions and Reference Modification. These can be combined to retrieve the date:

```
MOVE FUNCTION CURRENT-DATE(1:8) TO Date-8
```

3.3.2 Unisys

Unisys has implemented the 4-year extensions to the ACCEPT statement using the syntax from the proposed ANSI 97 standard. Both Unisys COBOL compilers at HUD were modified in April, 1997 to support the ANSI 97 current date retrieval functions. The extended ACCEPT is the simplest way to find a 4-digit current year.

Syntax: **`ACCEPT DATE-IN FROM`** date-clause.

Assuming the command is executed on Jan. 2, 1998 the results are:

Date Clause	Format	
DATE YYMMDD	YYMMDD	980102
DATE YYYYMMDD	YYYYMMDD	19980102

For Julian Dates:

Date Clause	Format	
DAY YYDDD	YYDDD	98002
DAY YYYYDDD	YYYYDDD	1998002

3.3.2.1 ASCII COBOL (ACOB)

The interim date routine, **`C$CURDATE`**, returns the ISO standard date **`(YYYYMMDD).`** For new code, or code being revised, the use of ACCEPT xxx FROM … is preferred. See **Section 3.3.2**.

```
CALL 'C$CURDATE' USING Date-8.
```

As an alternative, use the **`ACCEPT`** capability documented in **3.3.2**

3.3.2.2 UCS COBOL (UCOB)

UCS COBOL supports the new ANSI standard capabilities of Intrinsic Functions and Reference Modification. These can be combined to retrieve the date:

```
MOVE FUNCTION CURRENT-DATE(1:8) TO Date-8
```

As an alternative, use the **`ACCEPT`** capability documented in **3.3.2**

3.3.3 LAN

While mainframe code is generally written in COBOL, there are a variety of development tools on the LAN. 1-2-3, Access, Excel, Foxpro and other tools are used. The software companies who produce these tools are responding in a variety of ways to Year 2000 issues. For information about the changes needed for these products, contact the **Year 2000 Assistance Service** via cc:Mail at *team_2000@hud.gov*, or call (202) 755-2000.

3.4 Internal to a System

Dates used internal to a system may be passed program to program in any format, so long as the century is unambiguous (see description below; also, **see Section 4.1.2**, ***Windowing/Century Derivation/Coding)***. This may be accomplished through the use of windowing techniques.

An ambiguous century would be that of a birth date. A birth year of 95 could mean either 1895 or 1995. Windowing techniques will not work for dates spanning over 100 years.

Regardless of whether the system handles dates with 2-digit or 4-digit years, each program in the system must accept a system date with a 4-digit year. Once the program accepts the date, it can truncate the field down to a 2-digit year-field.

The panel recommends that new system development efforts use expanded dates in the format **CCYYMMDD** of numeric data type and proprietary database date fields with 4-digit years, such as the DB2 date data type.

3.5 Database Management Systems

Systems that share databases must have a shared standard for the database schema. The renovation teams responsible for these systems must work together with system DBA's to mutually decide upon a set of standards. For example, the teams must decide whether they will expand database fields, change the field names, migrate to a new version, and/or use compression to store expanded dates. Once agreed upon, the new schema must be made available to all relevant systems.

The panel recommends that new systems use DBMS defined date formats with 4-digit years.

3.6 On-line Screens

Dates displayed on screens shall continue to be displayed in 2-digit year format unless it is critical that the century be displayed. This may be the case if the century is ambiguous.

Dates accepted on screens shall continue to be accepted in 2-digit year format unless a century cannot be derived successfully from the year through a common date routine, and therefore, it is ambiguous.

3.7 Reports

Existing reports shall continue to display dates in 2-digit year format. The date can be handled in an expanded format, but simply converted to a 2-digit year format for output to the report.

3.8 Bridges

All bridges shall be either separate program components or separate callable modules so that they may be easily removed once a sending or receiving application moves to year 2000 compliance. In situations where an architecture prohibits the use of a separate program component, such as in the case of some on-line or real time programs, bridges may be built into a program unit.

We advise making such internal bridging logic as modularized as possible (preferably a callable module) so that it can be easily removed if necessary. It is recommended that these bridges be well documented.

The team that moves an application to expanded date formats is responsible for building bridges to and from applications with which it communicates that do not have expanded dates.

All bridges must use standard conversion or windowing routines from the TransCentury date routines.

New systems are responsible for creating bridges to and from existing systems, if necessary.

4.0 Approaches for Renovation

Approaches for renovation shall be decided at the application level. However, it is important to remember that application development teams who are accessing a shared DBMS need to come to a mutual understanding of how the DBMS will be made year 2000 compliant.

It is also important to note that a date field or variable needs renovation only if that field or variable is used for some type of computation, conditional statement, or sort where the century will be relevant. For example, if a 6-digit date field is used only to display the date on a report, then there is no need to renovate the date at all. This does not apply to system date. System date must always be accepted from a source that returns a 4-digit year.

At the very least, it is critical that a renovation team does a basic cost/benefit analysis of the various approaches. After reading the descriptions of the approaches and their benefits and drawbacks, outline those benefits and drawbacks that apply to your system and compare the two. Even basic estimates of costs and impacts specific to your system will help make the decision more informed and objective.

Whichever approach an application renovation team employs, it must be communicated to the Team 2000 Project Office and entered into the STATUS 2000 database. STATUS 2000 is a Lotus NOTES® application, available to renovation team members, which serves as the central location for information relating to Year 2000 systems and schedules (See **STATUS 2000 Database Information**, **Document I**, in the **Reference Library** for user information). The information to be entered must include the System Code, points-of-contact, the system acronym, and the selected approach, among other items.

4.1 Description of Approaches

4.1.1 Field Expansion

The *field expansion* approach expands the Year field from 2 to 4 bytes to include the century. Format YYMMDD will be changed to CCYYMMDD while the Julian format YYDDD will become CCYYDDD. For example, 951001 will be 19951001, 95365 will be 1995365.

This is a straightforward solution that will result in a maintainable system. However, this will require numerous database changes that will adversely impact the volume of coding modifications and testing.

Consider the following:

Cost of Development of File/Data Conversion Programs

If a date field is expanded, data or file conversion flows and programs must be planned and designed. Development of conversion programs for flat files and database files can add significantly to the cost and time of the Year 2000 Implementation.

Installation

Every expansion change would add cost to file conversion and increase the difficulty of conversion. If the file is large and is used by a 7-day by 24-hour application, it may be difficult to find a time when the conversion can be scheduled to run.

Scheduling Impact

When you change a file format, you must change all programs that read or write to that file. This makes the conversion more costly and the timing more difficult. For a frequently accessed file, this could be a very serious problem. If file formats are kept unchanged, then each program can be modified separately from the others.

Cost of Development of Bridge Programs

Problems with scheduling can be reduced by developing bridge programs which convert between the old and new file formats. There is, however, a cost in developing bridge programs.

Impact to Existing Test Data

Test data (such as test scripts, test data models) that are being used must be updated to the new format for them to work with the new version of the application.

Impact to Archived Data

If you need to recover data from an archived file, the new software will not work with it.

Other Technical/Operational Constraints

In older (non-relational) systems, a change to an index or to the width or data type of an existing column is an operation not to be undertaken lightly. Typically it involves bringing the entire system to a halt. Many installations simply cannot afford to bring the system to a halt. Such operational and/or technical restrictions should be taken into account. In this case, migrating the system to a newer architecture might be warranted.

File Record Size/Table Space/Index Size

The record size might already be at a maximum value (for example, file record size is 512 bytes) and adding two bytes is a problem (for example, file size exceeding VSAM maximum of 4.3 billion bytes). Also, increasing the length of a date column has direct impact to storage and might require constant tracking of disk space usage. Expansion of widely used files and tables would require changes in JCL and DYL, and has an impact to storage (such as a bigger disk space requirement) and performance (for example, I/O access, deadlocks, time-outs). On the UNISYS this may mean increasing the number of tracks assigned to a file in the ECL. Renovation teams must estimate the impact of expanded data fields in advance of a renovation effort in order to gauge the effect on production runs. Additional purchases of storage or machines must take place prior to renovation.

Data Entry

Typing in a 4-digit year takes more keystrokes than typing in a 2-digit year. Expanding would require additional training for users and data encoders. Test scripts must be updated with new screens for them to work. Therefore, do not expand date fields that are being entered, unless the century of the date is ambiguous (see **Section 4.1.2**, ***Windowing/Century Derivation/Coding***), such as in date of birth.

Screen and Report Layout

Field positioning and filler fields have to be adjusted. Columnar formats should be maintained. The line may already be full, requiring significant movements in layout. Such changes could possibly affect program logic apart from the variable declarations. Also, there would be user impact and training to consider if changes are major. When variables are painted on screens or windows, whether for entry or display, their formats should remain the same (such as a 2-digit year). As for reports, dates that are for viewing purposes only (for example, AS OF date from page headings or footers) should remain as is.

4.1.2 Windowing/Century Derivation/Coding

This approach retains the 2-digit date field, but modifies the application so that it will treat low year-field values as 21st century dates. A cut-off year must be selected and the code modified so that the application will treat any year-field value that is less than the value of the cut-off year as a year in the 21st century. For example, if we determine that a certain variable, say a policy expiration date, can never have a date earlier than 1956, we write the code to assume that all policy expiration dates that are less than or equal to 55 are in the 21st century. An application may need to employ more than one cut-off year, depending upon functional requirements.

The above technique is more specifically called the *fixed window technique*, since it is based upon a fixed year. For example, if the cut-off date is 50, then all dates less than 50 belong to the 21st century, (for example., 2000 through 2049) and all dates greater than or equal to 50 belong to the 20th century, (for example, 1950 through 1999.) In this example, the Year 2000 problem is essentially converted to the Year 2050 problem.

For the *sliding window technique*, the century of a date will depend on the number of years in the past or future relative to the current year. For instance, if the range is 30 years in the past and 69 years in the future, based on the current year of 1995, then the date window is 1965 through 2064. Next year, 1996, the window advances to include 1966 through 2065. This technique provides a permanent solution to the Year 2000 Problem.

Windowing approaches cannot always be applied, however. In some cases, a date variable can represent years that are more than one century apart. We cannot define a cut-off date in those cases.

To identify a date variable's date range we must determine the difference between the earliest possible value that it may have and the latest possible value that it may have. For example, if the variable represents “next birthday,” then the range is exactly one year. (The earliest possible value is tomorrow, and the latest possible value is one year from today.) On the other hand, if the variable represents the birth date of a resident of this town, then the date range can be over 100 years. (The earliest possible value is the birth date of the oldest resident, who could be over 100 years old; and the latest possible value is today.) So a birth year of 95 could mean either 1895 or 1995. Windowing techniques will not work for dates spanning over 100 years.

The windowing approach saves time and costs up front, but leaves a complicated coding solution for future maintenance and support. The coding implementation is also more error prone.

Consider the following:

Complexity of logic

The application logic becomes more complex due to the windowing logic necessary for each impacted date related element. Due to the complexity of the logic, ongoing maintenance will be more challenging and more expensive.

Dates that span 100 years

2-digit year data spanning 100 years is not possible. For instance, this is not an option for birth dates. (See the explanation above.) Bridging requirements grow with time. As more and more interfacing systems carry 4-digit years, legacy applications will have to build bridges in order to exchange dates in a 4-digit year format. This need will increase over time.

Sorting is complex Sorting on years without a century is not possible once the data crosses the Year 2000 boundary. Some dates in key fields may need to be expanded selectively, therefore increasing the complexity of this operation.

4.1.3 Compression/Encoding

This approach employs any of a number of schemes to compress or encode a century into a date variable without expanding the fields of databases, data sets, and other data structures. An example is to use a third unused digit to represent the century, or to use hexadecimal encoding instead of decimal.

Some examples of encoding:

A0 (Alpha) Approach Any year after 1999 is represented by a letter in the first digit and a number in the second. The letter indicates the decade in the 21st century, beginning with A as the first decade (2000), B as the second (2010), and so on. The number indicates the number of years added to the decade.

Continuous Century Indicator A single digit on/off switch is added to the beginning of the existing two-digit year field. This digit indicates either the 20th or 21st century (0 or 1 respectively). This approach requires that the date field be redefined from numeric to alphanumeric and expanded from two to three bytes.

Non-Continuous Century Indicator A single digit on/off switch is added to the record, separate from the two-digit numeric field. The digit indicates either the 20th or 21st century (0 or 1 respectively). This approach does not require that the date field be redefined from numeric to alphanumeric. However, it does require an additional byte in the data record.

Offset Date The entire date is represented as the number of days elapsed since January 1, 1850 (or some other established year). This approach requires that the date field be a 5-digit number represented by a three-byte, packed field. This approach may also be known as a Lilian year.

Hex-Year Hexadecimal representation of the year is based upon a base-16 numeric system (as opposed to our base-10 decimal system). Hex offset from base year of 1900 is also called 1-byte binary.

Hex-Alpha (Also known as the Zoned Decimal approach) Four characters represent the years through 1999 by numerically indicating the century and decade. Years after 2000 use an alphanumeric character to indicate the decade while a fourth decimal character represents the year from zero to nine.

Alternative Hex-Year A year in the 21st century will be indicated as **xFA** plus the number of years after 2000 stored in hex. 20th century dates will continue to be coded as two-byte years. The two-byte year field is maintained and dates will sort properly according to their hex values.

YYYDDD A 6-character date is represented by four bytes packed. YYY is the number of years since 1800 and DDD is the Julian day.

For relative date usage, HUD has chosen 1/1/1601, the ANSI standard, as its standard.

This method is generally viewed as the least desirable.

Consider the following:

1. It requires changes to both data and programs.
2. Renovation teams must simultaneously convert all applications using the data.
3. Encoded dates require conversion whenever you work with them, therefore calls to and processing in encoding modules will slow down performance.
4. Encoded data is impractical for human reading or printing (raw data).
5. Encoded dates require conversion when bridging to other applications. (Comp field dates will have a problem if the character set changed during conversion.)
6. It may be difficult to index on the compressed data element.

4.2 Recommendations on Renovation Approaches for Legacy Applications

In general, of the above approaches, field expansion is the most desirable, and compression is the least desirable; however, the variety of legacy applications in operation at HUD does not permit the panel to recommend a single approach that will work for all.

The panel does provide the following two approaches, the Field Expansion Approach and the Windowing & Expansion Approach for legacy applications. The panel does not recommend the compression approach.

Originally Published on January 16, 1997; Revised November 19, 1997

4.2.1 The Field Expansion Approach for Legacy Applications

1. Field expansion is used for all date elements that are determined to be impacted by the Year 2000 problem.

Expand 2-digit year date fields defined in working storage, linkage section, and copybooks to 4-digit year variables.

In addition to these specific date variables, some fields, such as save areas and other storage variables that can include dates should also be inspected. These fields should be expanded by two for each embedded date.

For example:

```
01  WS-FILE1-AREA.
    05  WS-TEMP-FILE1-ACCT-NO             PIC
X(6).
    05  WS-TEMP- FILE 1-ACCT-NM           PIC
X(30).
    05  WS-TEMP- FILE 1-ACCT-OPEN-DT      PIC
X(8).
    05  WS-TEMP- FILE 1-ACCT-CLOSE-DT     PIC
X(8).
01  WS-TEMP-FILE1                         PIC
X(52).
    * WS-TEMP- FILE1 IS A SAVE AREA OF WS-
FILE1-AREA*
```

would be changed to

```
01  WS-FILE1-AREA.
    05  WS-TEMP-FILE1-ACCT-NO             PIC
X(6).
    05  WS-TEMP- FILE 1-ACCT-NM           PIC
X(30).
    05  WS-TEMP- FILE 1-ACCT-OPEN-DT      PIC
X(10).
    05  WS-TEMP- FILE 1-ACCT-CLOSE-DT     PIC
X(10).
01  WS-TEMP-FILE1                         PIC
X(56).
```

Note that the size of WS-TEMP-FILE1 must be changed, also, from 52 to 56.

For COBOL, many copybooks have FILLERs at the end. If the FILLER is sufficiently large to absorb the increased size of the date fields, it should be reduced by the amount of additional bytes added to the copybooks due to the expansion of the date fields.

Modifications to the logic within the program that reference these declarations may be minimal, since all internal date fields would also be expanded. Thus, comparisons and calculations between two internal fields would still work.

The advantages of this approach are that it can provide 4-digit year format, and it minimizes future risk of century related problems. For this approach, the system might experience a performance impact due to increased time in processing and date access.

2. Remove all existing century derivation logic internal to programs.

Century derivation logic is one type of an ad-hoc solution. Such solutions may be complex and are a maintenance problem. If the years have been expanded, then the century derivation logic should be removed from existing code.

While this approach minimizes complexity, increases maintainability, and decreases potential sources of date related problems, it will require modification of code that may already be working correctly.

3. Use a Standard Conversion Subroutine for Converting Dates

Always use a standard conversion subroutine for converting dates.

When an application accesses a variable that is external to it, and expansion of that variable is not possible, then a windowing technique is required. This approach requires changes to programs only, to determine the century of a 2-digit year. Rather than changing each and every module that uses 2-digit externals, a common routine shall be used (see information on routines above).

When a 2-digit external variable is read into or written from a 4-digit internal variable, change the module that performs the I/O read and/or write. Call the conversion subroutine after the I/O call for a read or drop the century before the I/O call for a write.

For this rule, a Year 2000 solution is provided for those 2-digit year variables that will not be expanded (for example, from files, screens, databases). Also, there is increased reusability and maintainability. Expect a performance impact due to the overhead of 2 to 4-digit year conversion, although by some estimates this overhead is usually trivial. Another disadvantage is the need to define a process on how the relative cutoff dates for each date variable may be defined and updated (see the **Section 4.1.2**, ***Windowing/Century Derivation/Coding***).

4. Always Use a Standard Conversion Subroutine for Leap Year Calculations

Always use a standard subroutine for leap year calculations.

Modify all programs with leap year calculations to call a standard leap year routine. This is regardless of whether the inline leap year calculation uses 2-digit or 4-digit year variables. Making the leap year calculation a common routine ensures correctness and eliminates repetitive testing of said logic in every module.

This technique requires a need to change module(s) to use the subroutine in all cases.

5. Change all hard coded date values to date variables/literals with a century field.

Examine all hard coded dates used within program including date literals that were defined to only have a YY value. Make sure the literal values are Year 2000 compliant.

Hard Coding Within Program

Example:

```
IF INPUT-TXN-DT < 80-09-21
     MOVE MSG-TXN-DT-TOO-EARLY TO SCREEN-TXN-DT-S
END-IF
```

would be changed to

```
IF INPUT-TXN-DT-D < LT-MIN-ACCT-OPEN-DT
     MOVE MSG-TXN-DT-TOO-EARLY TO SCREEN-TXN-DT-S
END-IF
```

Where **`LT-MIN-ACCT-OPEN-DT`** is defined **`as PIC X(10) VALUE 1980-09-21`** .

Date Literals

Example:

```
05   LT-FIRST-FYE-DT     PIC X(8) VALUE 81-08-31
```

would be changed to

```
05   LT-FIRST-FYE-DT     PIC X(10) VALUE
1981-08-31.
```

Look out for hard-coded or literal dates that were used as default date values (for example, minimum, maximum). If applicable, nulls, low values, or high values should be used as the default value instead of any specified date. A particularly dangerous example is if one of 99, 99365, 99/12/31 or 00 is used to indicate an invalid date.

Please report widespread use of any such default or special dates to the Year 2000 Technical Panel via the TEAM 2000 cc:Mail address. The Year 2000 Project will serve as a central source of Year 2000 specific technical information for the HUD IT community.

6. Conversion of date variables to standard formats is not required within an application

Date format standards exist that were set by external organizations, such as., ANSI X3.30, FIPS 4-1, ISO 8601. No other changes should be made to the date format used by the application, other than the year expansion. Converting all existing date variables to use standard formats would have a large impact to the project cost and complexity.

Also, this would require additional scenarios for testing before conversion. On the other hand, the resulting code would be much more consistent and more maintainable.

7. External modules are not required to use the standard date library

A standard date routine has been purchased that includes hundreds of date calculations, but rewriting existing date calculation modules to call the new standard date routines is not required. To convert other modules to use the library would require the cost of modification of presumably correct code, for the benefit of more modular code, and at the risk of conversion mistakes. Note that this rule is only applicable for old code and should not be used for any new code being added to a legacy application.

8. Screen and Report Layout

Do not expand fields on screens or reports unless necessary (see date range in **Section 4.1.2**). Field positioning and filler fields have to be adjusted. Columnar formats should be maintained. The line may already be full, requiring significant movements in layout. Such changes could possibly affect program logic apart from the variable declarations. Also, there would be user impact and training to consider if changes are major.

When variables are painted on screens or windows, whether for entry or display, their formats should remain the same (for example, 2-digit year). As for reports, dates that are for viewing purposes only (such as, AS OF date from page headings or footers) should remain as is.

9. Data Entry

Typing in a 4-digit year takes more keystrokes than typing in a 2-digit year. Expanding would require additional training for users and data encoders. Test scripts must be updated with new screens for them to work. Therefore, do not expand date fields that are being entered unless it is necessary due to an ambiguous century (see date range in **Section 4.1.2**).

See also **Sections 5-7**. These sections provide some additional guidance based on platform and language.

4.2.2 The Windowing & Expansion Approach for Legacy Applications

This approach for legacy systems combines both the field expansion and windowing approaches. Field expansion is used internal to programs (computational logic, conditional statements, and so on). Windowing is used for external variables (data files, databases). This approach maximizes the benefit of working with 4-digit years in our program logic, and minimizes the cost of data conversion and file and database expansion.

See also **Sections 5-7**. These sections provide some additional guidance based on platform and language.

4.2.2.1 Internal Variables

1. Expand 2-digit year variables to 4-digit

All 2-digit year date fields defined in working storage, linkage section, and copybooks that are not external to the application shall be expanded to 4-digit year variables.

In addition to these specific date variables, some fields, such as save areas and other storage variables that can include dates should also be inspected. These fields shall be expanded by two for each embedded date.

For example:

```
01  WS-FILE1-AREA.
    05 WS-TEMP-FILE1-ACCT-NO                    PIC
X(6).
    05 WS-TEMP-FILE 1-ACCT-NM                   PIC
X(30).
    05 WS-TEMP- FILE 1-ACCT-OPEN-DT             PIC
X(8).
    05 WS-TEMP- FILE 1-ACCT-CLOSE-DT            PIC
X(8).
01  WS-TEMP-FILE1                               PIC
X(52).
* WS-TEMP- FILE1 IS A SAVE AREA OF WS- FILE1-AREA*
```

would be changed to

```
01  WS-FILE1-AREA.
    05 WS-TEMP-FILE1-ACCT-NO                  PIC X(6).
    05 WS-TEMP- FILE 1-ACCT-NM                PIC
X(30).
    05 WS-TEMP- FILE 1-ACCT-OPEN-DT           PIC
X(10).
    05 WS-TEMP- FILE 1-ACCT-CLOSE-DT          PIC
X(10).
01  WS-TEMP-FILE1                             PIC
X(56).
```

Note that the size of WS-TEMP-FILE1 must be changed also from 52 to 56.

For COBOL, many copybooks have FILLERs at the end. If the FILLER is sufficiently large to absorb the increased size of the date fields, it should be reduced by the amount of additional bytes added to the copybooks due to the expansion of the date fields.

Modifications to the logic within the program that reference these declarations may be minimal, since all internal date fields would also be expanded. Thus, comparisons and calculations between two internal fields would still work.

Special attention should be given when program logic involves a combination of internal and external variables which are in 2 digit format. The implementation should have 4 digit working storage copies of the external variables, and the procedural code should reference the working storage copies, not the external variables themselves.

The advantages of this approach are that it can provide 4-digit year format, and it minimizes future risk of century related problems. For this approach, the system might experience a performance impact due to increased time in processing and date access.

2. Remove all existing century derivation logic internal to programs.

Existing century derivation logic is one type of an ad-hoc solution. Such solutions may be complex and are a maintenance problem. If the years have been expanded, then the century derivation logic should be removed from existing code.

While this approach minimizes complexity, increases maintainability, and decreases potential sources of date related problems, it will require modification of code that may already be working correctly.

3. Always use a standard conversion subroutine for external variables.

When an application accesses a variable that is external to it, and a data change is not possible, then a windowing technique is required. This approach requires changes to programs only, to determine the century of a 2-digit year. Rather than changing each and every module that uses 2-digit externals, a common routine shall be used (see information on routines above).

When a 2-digit external variable is read into or written from a 4-digit internal variable, change the module that performs the I/O read and/or write. Call the conversion subroutine after the I/O call for a read or drop the century before the I/O call for a write.

For this rule, a Year 2000 solution is provided for those 2-digit year variables that will not be expanded (for example, from files, screens, databases). Also, there is increased reusability and maintainability. Expect a performance impact due to the overhead of 2 to 4-digit year conversion, although by some estimates the added overhead is usually trivial. Another disadvantage is the need to define a process on how the relative cutoff dates for each date variable may be defined and updated (see the **Section 4.1.2**, ***Windowing/Century Derivation/Coding***).

4. Always use a standard subroutine for leap year calculations.

Modify all programs with leap year calculations to call a standard leap year routine. This is regardless of whether the inline leap year calculation uses 2-digit or 4-digit year variables. Making the leap year calculation a common routine ensures correctness and eliminates repetitive testing of said logic in every module.

This technique requires a need to change module(s) to use the subroutine in all cases.

5. Change all hard coded date values to date variables/literals with a century field.

Examine all hard coded dates used within program including date literals that were defined to only have a YY value. Make sure the literal values are Year 2000 compliant.

Hard Coding Within Program

Example:

```
IF INPUT-TXN-DT < 80-09-21
       MOVE MSG-TXN-DT-TOO-EARLY TO
SCREEN-TXN-DT-S
END-IF.
```

would be changed to

```
IF INPUT-TXN-DT-D < LT-MIN-ACCT-OPEN-DT
       MOVE MSG-TXN-DT-TOO-EARLY TO
SCREEN-TXN-DT-S
END-IF.
```

Where `LT-MIN-ACCT-OPEN-DT` is defined `as PIC X(10) VALUE 1980-09-21.`

Date Literals

Example:

```
05   LT-FIRST-FYE-DT PIC X(8) VALUE 81-08-31
```

would be changed to

```
05   LT-FIRST-FYE-DT PIC X(10) VALUE 1981-08-31.
```

Look out for hard-coded or literal dates that were used as default date values (for example, minimum or maximum). If applicable, nulls, low values or high values should be used as the default value instead of any specified date. A particularly dangerous example is if one of 99, 99365, 99/12/31 or 00 is used to indicate an invalid date.

6. Conversion of date variables to standard formats is not required

There exist date format standards that were set by external organizations, for example, ANSI X3.30, FIPS 4-1, ISO 8601. No other changes should be made to the date format used by the application, other than those requiring year expansion. Converting all existing date variables to use standard formats would have a large impact to the project cost and complexity. Also, this would require additional scenarios for testing before conversion. On the other hand, the resulting code would be much more consistent and more maintainable.

See also **Sections 5-7**. These sections provide some additional guidance based on platform and language.

4.2.2.2 Interface variables

A 4-digit year is always better since it has a more maintainable design than a 2-digit year. Therefore, if some or all external variables are converted from 2-digit to 4-digit representation, the maintainability will be improved, but at the expense of higher conversion cost. Considering this trade-off, Year 2000 conversions for legacy applications may decide to minimize expansion changes to existing interface variables.

Changing fields in files and databases is expensive, due to installation dependencies, file conversion costs, bridge program costs, and problems with archive data. Expanding screen and report fields is also expensive, since there will be a need to repaint screens, regenerate layouts and update test files and scripts.

Conversion to a standard format is also not required. Date formats set by external organizations, like ANSI X3.30, FIPS 4-1, ISO 8601 could be selected by a renovation team as its standard format, but external date variables of legacy applications with a format different from that of the standard are not required to comply. Besides year expansion of external date variables, there are no other required changes. Converting all date interfaces to use standard formats has a direct impact to the project cost and complexity. Also, this would require additional scenarios for testing before conversion.

Windowing is employed when writing to and reading from external date elements. One sure exception to this rule is variables that have a date range of 100 years or more. Another is variables that are used for sorting and indexing, where these operations cannot be customized to handle century derivation (for example, DBMS or File Management software index sequencing during record inserts). For such cases, there is no other option but to expand to 4-digit, unless, of course such expansion is not technically possible.

Below are the different factors to consider when making choices about expanding external variables:

1. Screen and Report Layout

Do not expand fields on screens or reports unless necessary (see date range in **Section 4.1.2**). Field positioning and filler fields have to be adjusted. Columnar formats should be maintained. The line may already be full, requiring significant movements in layout. Such changes could possibly affect program logic apart from the variable declarations. Also, there would be user impact and training to consider if changes are major.

When variables are painted on screens or windows, whether for entry or display, their formats should remain the same (for example, 2-digit year). As for reports, dates that are for viewing purposes only (for example, AS OF date from page headings or footers) should remain as is.

2. Data Entry

Typing in a 4-digit year takes more keystrokes than typing in a 2-digit year. Expanding would require additional training for users and data encoders. Test scripts must be updated with new screens for them to work. Therefore, do not expand date fields that are being entered unless it is necessary due to an ambiguous century (see date range in **Section 4.1.2**).

3. External modules are not required to use the standard date library.

A standard date routine has been purchased that includes hundreds of date calculations, but rewriting existing date calculation modules to call the new standard date routines is not required. To convert other modules to use the library would require the cost of modification of presumably correct code, for the benefit of more modular code, and at the risk of conversion mistakes. Note that this rule is only applicable for old code and should not be used for any new code being added to a legacy application.

4.3 Recommendations on Approaches for New Applications

4.3.1 Internal Variables

All date fields/variables shall be of 4-digit year, specifically in the format CCYYMMDD.

4.3.2 Interface Rules for New Applications (and New Code Added to Legacy Applications)

All interface variables have 4-digit years.

Date variables for the application must always be defined to contain a 4-digit year. Exceptions are those cases where there is a set limitation (for example, maximum screen size, maximum report length).

Data element attributes such as data type, length, structure, report format, internal format shall always take the format of a 4-digit year. The only attribute that may contain a 2-digit year structure is the screen or window format. This is to save on keystrokes when entering date values to screen.

For some on-line architectures, like MES (Install/1) and MFS (IBM), it is allowed for an element to have a display format that is different from the internal format. For instance, the application displays dates consistently in a calendar (MM/DD/YY) format and passes them to another application in a Julian-7 (CCYYDDD) format. The conversion to the new format is done by the technical architecture automatically. Note that the default century used by the architecture is based on a system parameter which serves as a cut-off year. This value was set by the user during installation. This value should be checked over a period of time to verify correctness in defaulting.

If on-line architecture does not support the above feature, then application architecture itself should perform the conversion of the screen field to the standard format for internal use.

When designing hardcopy reports, be careful using 2-digit years. Any such data can be (and often is) added to a data set and then read by another program. Thus, such data should not be considered as a non- impact item and acceptable to have a 2-digit year. A usual exception to this is run-date displayed from page headers and footers; however, any run-date should be moved from a system date with a 4-byte year.

When new code interfaces with legacy files, 2-digit external variables may have to be read from or written into 4-digit variables. In this case, make sure that no other code deals with the 2-digit representation.

See **Sections 5-7** for more guidelines based on platform.

4.4 Summary of Benefits and Drawbacks to Each Approach

4.4.1 Expansion

The field expansion approach has the following benefits:

1. It is a higher quality fix, meaning that once you have made the fix, you will not have to revisit the problem again.
2. It is not as code intensive as the windowing approach; program complexity should not increase.
3. It is good for systems needing major renovation or overhaul.
4. It reduces bridging requirements as more and more systems use 4-digit year industry standard.
5. It enables consistency of approach with newly developed century compliant systems.

The field expansion approach has the following drawbacks:

1. It still requires some logic change:
   - ➤ Data declarations at the very least
   - ➤ Logic moving 4 digit years to 2 digit data items
   - ➤ Interface bridges receiving 2 digit years still need logic to infer the century.
2. Changing data structures impacts the whole system (programs, copybooks, jobs, VSAM defines, database schema).
3. Every recompiled program must be tested.
4. Increases space required and may impact processing volumes and speeds.
5. Increases complexity of configuration management and parallel production enhancements.
6. All data must be converted at same time as code conversion.
7. Historical data will not be compatible (archived tapes).

4.4.2 Windowing/Century Derivation

Windowing or century derivation has the following benefits:

1. It is less time consuming and less expensive in the short term.
2. It is good for smaller projects.
3. It is good for stable/reliable systems or those with little interfacing.
4. Program changes can be made and tested in isolation from other interdependent systems.
5. Risk and exposure to change is limited to parts of system where logic is modified.
6. Testing can be targeted, reducing testing time and cost.
7. Configuration management is simplified.

Windowing has several drawbacks:

1. Logic is more complex and ongoing maintenance more expensive.
2. 2-digit year data spanning 100 years is not possible. For instance, this is not an option for birth dates.
3. As more and more interfacing systems carry 4 digit years, bridging requirements increase.
4. Sort complexity may be increased, some dates in key fields may need to be expanded selectively.

4.4.3 Compression

Field compression has the following benefits:

1. There may be no need to expand data fields or rebuild databases.
2. It saves data storage space.
3. It may force strange codes out of the data (text such as *N/A* in date fields.)

Drawbacks Field compression has the following drawbacks:

1. It requires changes to both data and programs.
2. Renovation teams must simultaneously convert all applications using the data.
3. Encoded dates require conversion whenever you work with them, therefore calls to and processing in encoding modules will slow down performance.
4. Encoded data is impractical for human reading or printing (raw data).
5. Encoded dates require conversion when bridging to other applications (Comp field dates will have a problem if the character set changed during conversion).
6. It may be difficult to index on the compressed data element.

5.0 UNISYS Specific Issues

5.1 DMS1100/2200

Some DMS1100/2200 applications have adopted the following standard for date field expansion:

```
01 ORIGINAL-FIELD-8
05 FIELD-CENTURY        PIC 9(2)
05 ORIGINAL-FIELD       PIC 9(6)
```

where **ORIGINAL-FIELD** is the original field name. This technique allow continued use of the **ORIGINAL-FIELD** element, yet allows use of **ORIGINAL-FIELD-8** when the 8-digit length is required.

The Year 2000 Project has ordered a tool to aid with Year 2000 conversions in these systems. The tool is called I-QU 2000 and it is offered by KMSystems. I-QU 2000 identifies dates in a DMS2200 database structure and automatically generates the code necessary to perform the required conversions.

The tool also provides reports that assist DBA's in determining the best plan for converting specific areas, records and programs.

Some DMS1100/2200 application teams have employed compression techniques to avoid expanding fields in the database. Instead of storing data as a PIC 9(6) DISPLAY, the data is stored as PIC 9(8) COMP. Since field expansion is a very large issue in DMS1100/2200 databases, this is an option, although it should be employed only as a last resort.

Remember, employing this technique has the following drawbacks:

1. It requires changes to both data and programs.
2. Renovation teams must simultaneously convert all applications using the data.
3. Encoded dates require conversion whenever you work with them, therefore calls to and processing in encoding modules will slow down performance.
4. Encoded data is impractical for human reading or printing (raw data).
5. Encoded dates require conversion when bridging to other applications (Comp field dates will have a problem if the character set changed during conversion).
6. It may be difficult to index on the compressed data element

The Technical Panel does not recommend employing compression.

See also the document HUD Year 2000 Platform Issues.

5.2 LINC

Migration to LINC Ver. 16 is recommended but not required.

Since LINC 16 is Year 2000 compliant, applications processing LINC date functions do not need to use date functions from the TransCentury date routines. LINC date functions are also acceptable. LINC date functions are Year 2000 compliant. LINC applications must accept a system date that returns a 4-digit year, like any other UNISYS application.

6.0 Hitachi

See **Section 2.5.1**, in Chapter 2, ***Application Analysis***.

7.0 PC/LAN

See **Section 2.5.3**, in Chapter 2, ***Application Analysis***.

8.0 Year 2000 Compliance

8.1 Overview

HUD's business rules will be computed accurately before, during, and after January 1, 2000, regardless of whether the dates were received from the computer's operating system, from a data repository, or from an external source (interface or user input).

8.2 Testing for Year 2000 Compliance

(See also the Year 2000 Testing Strategy document)

An application development team will create baseline test results based on a thorough set of test conditions prior to renovation. Authors of test cases must pay special attention to date logic, although not limit their test cases to date logic. Applications shall be able to produce matching baselines after Year 2000 renovation, since renovation should not involve any functional change to the system.

In addition, the following test cycles must be applied:

Expected results are achieved when test data is aged to December 31, 1999, January 1, 2000, and beyond.

Expected results are achieved before, during, and after the software crosses the century boundary (system date transition from before December 31, 1999 to after January 1, 2000).

Expected results are achieved when leap year/non-leap year dates are used. For example, system date of 2/29/1996, 2/29/2000; test data date of 2/29/2000; finding day of the week for 3/1/2000. For non-leap year dates, what day of the week is after 2/28/2001; what date is 1 day after 2/28/2001?

Expected results are achieved when invalid dates are applied. For example, 4/31/2001, 2/29/2001, xx/xx/19xx, /40/0000.

Date simulation software or techniques will be required to execute some of the above tests.

8.3 Compliance Checklist from Standards Perspective

Yes	N/A	Item
		All calls for system date return 4-digit year (through TransCentury standard date routine or call to compliant O/S).
		Inter-agency data exchange uses 4-digit years.
		Inter-system data exchange uses 4-digit years unless a system meets the exceptions detailed above.
		Inter-system data exchange has been decided upon by both sending and receiving systems.
		Requests for new exceptions to the standards have been sent via cc:Mail to TEAM 2000.
		All impacted dates have been identified and addressed.
		Renovation approach has been identified to TEAM 2000 via cc:Mail.
		Application has created bridges to and from other systems as necessary.
		Bridges use conversion routines from TransCentury.
		Applications that use common database have coordinated with each other and DBA's to decide on common approach towards the database.
		Estimate costs/benefits before approach is chosen.
		Estimate impact on storage and performance estimated before approach is chosen.

8.4 Suggested Preparation for Test Phase

Using the Year 2000 Standards and Approach document, create a program checklist specifying detailed rules and exceptions for the selected approach. Include samples of variables, usage (for example, special date situations like age computation, sorts, comparisons, unique number generation, leap years), functions, formats, logic, and required steps/changes to be done specific to the Year 2000 Project. Create a corresponding test condition checklist for the changes indicated in the program checklist. This will be valuable when preparing test conditions for unit and string tests.

8.5 Certification

While Year 2000 compliance certification will be required of all systems, application development teams do not have to wait until certification is complete before migrating applications or parts of applications to the production environment.

Those systems that are thought to be Year 2000 safe and did not go through the renovation process will still be required to undergo Year 2000 certification.

See the document HUD Year 2000 Certification Strategy for more information on certification.

9.0 Naming Conventions

There will be no Year 2000 specific date naming conventions.

To convert each and every data variable to follow naming standards would require a large effort and cost, from defining data elements to modifying and testing code to use new names. Keep in mind that the deadline for the Year 2000 problem is non-negotiable, and HUD cannot afford to spend years doing variable conversions.

Individual renovation teams may decide to employ special naming conventions to denote and track their expanded or windowed dates. If that decision is made, it should be noted that HUD has naming conventions in place for dates. There are rules and guidelines written relating to data standardization. **Chapter 7** in the **Data Administration Standards and Procedures** (December 1994), spells out the rules and guidelines. **Section 7.2.2** in that document describes how to name data elements.

All data elements pertaining to dates:

- ➤ must contain the class word DATE or its abbreviation DT
- ➤ the class word must be the right-most word in the name

Examples:

```
BIRTH_DATE (not DATE_OF_BIRTH) or BIRTH_DT
BEGINNING_AMORTIZATION_DATE or BEG_AMORT_DT
```

For copies of the ***Data Administration Standards and Procedures***, contact the Central Information Management, Data Administration team.

(This page has been left blank intentionally.)